华中科技大学计算机学院
《计算机通信与网络》实验报告

班级 CS2006 姓名 杨释钧 学号 U202015471

项目	Socket 编程（40%）	数据可靠传输协议设计（20%）	CPT 组网（20%）	平时成绩（20%）	总分
得分					

教师评语：

教师签名：

给分日期：

目　　录

实验二 数据可靠传输协议设计实验

1.1 环境

1.1.1 开发平台

处理器：AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz

操作系统：Windows10 21H2 64 位

机带 RAM：16.0GB(15.4GB 可用)

开发平台：Visual Studio 2019

开发语言：C++

1.1.2 运行平台

处理器：AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz

操作系统：Windows10 21H2 64 位

运行软件：Visual Studio 2019

1.2 系统功能需求

1.2.1 总体要求

1. 可靠运输层协议实验只考虑单向传输，即：只有发送方发生数据报文，接收方仅仅 接收报文并给出确认报文；

2. 要求实现具体协议时，指定编码报文序号的二进制位数（例如 3 位二进制编码报文 序号）以及窗口大小（例如大小为 4），报文段序号必须按照指定的二进制位数进行编码；

3. 代码实现不需要基于 Socket API，不需要利用多线程，不需要任何 UI 界面。

1.2.2 具体要求

第一级：

1. 正确实现 GBN 协议；
2. 能够打印出运行过程中滑动窗口的变化过程，并将日志重定向到文件中，便于分析滑动窗口移动的正确性。

第二级：

1. 正确实现 SR 协议；
2. 能够打印出运行过程中发送方和接收方滑动窗口的变化过程，并将日志重定向到文件中，便于分析滑动窗口移动的正确性。

第三级：

1. 基于 GBN 协议实现一个简化版的 TCP 协议，应当支持快速重传和超时重传，重传时只重传最早发送且没被确认的报文段；
2. 能够打印出运行过程中滑动窗口的变化过程，并将日志重定向到文件中，便于分析滑动窗口移动的正确性；
3. 能够在发生快速重传的时候输出相关日志，并将日志重定向到文件中，便于分析快速重传的正确性。

1.3 系统设计

1.3.1 GBN 协议的设计

GBN 全称为 Go-Back-N，允许发送方发送多个分组而不需要等待确认，但已发送未确认的分组数不能超过 N[1]。下面将分别对 GBN 协议的发送方和接收方进行讨论。

对于发送方，GBN 协议在分组首部用 k-比特字段表示分组序号，已被传输但是还未确认的分组的许可序号范围可以看作是一个在序号范围内大小为 N 的窗口中，其示意图如图 1.1 所示。图中的 base 表示最早发送的未确认的序号，nextseqnum 表示下一个代发分组的序号。

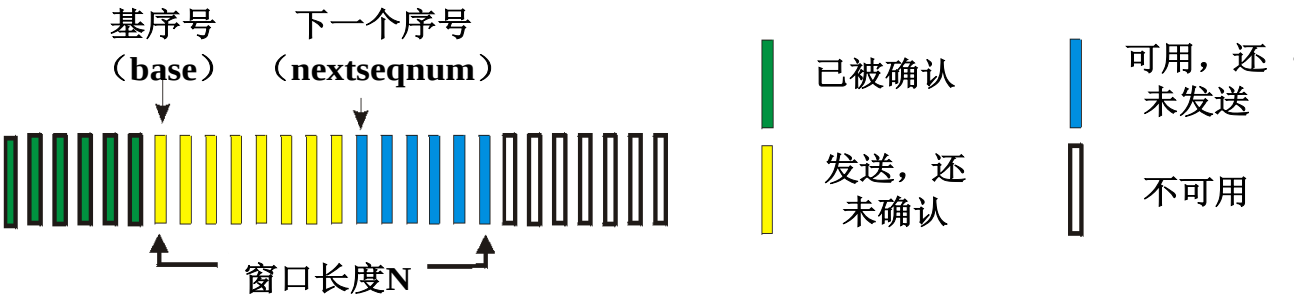

图 1.1 GBN 协议窗口示意图

容易知道，发送方需要对三个动作进行响应：

1. 当上层调用传递给发送方的时候，发送方需要提供 send 函数。首先检查当前窗口中是否还有可用的序号，如果没有的话说明窗口已满，否则的话就接受上层传递的数据，为其加上序列号，计算校验和，并且更新 nextseqnum。值得注意的是，如果发送分组的序号是 base 的话就启动计时器；
2. 当接收到接收方返回的 ACK 的时候，发送方需要提供 receive 函数。首先检查接收到的 ACK 是否损坏，如果没有损坏并且该 ACK 确认的序号大于等于 base 的话，就移动窗口，更新 base，重启计时器，否则忽略该信息；
3. 当当前的计时器超时时，发送方需要提供 timeoutHandler 函数，只需要把已发送但是还未被确认的分组重新发送一遍即可，也就是将区间[base,nextseqnum-1]中的分组全部发送一遍。

接收方只需要设计一个接受分组的函数 receive 即可，接收方本身维护了变量 ExpectNextPacketNum，意为期望收到的分组序号。如果接收方收到分组，首先检查分组校验和，如果没有损坏并且该分组的序号正好和 ExpectNextPacketNum 相等的话就将该分组传递给上层，并且给发送方回复一个对该序号的 ACK, 然后更新 ExpectNextPacketNum。否则丢弃该分组，发送 ExpectNextPacketNum 对应的 ACK。

1.3.2 SR 协议的设计

SR 协议是选择重传协议，和 GBN 协议相比，SR 协议在发送方也维护了一个窗口，接收方逐个对所有正确收到的分组进行确认，而不是使用累计确认，从而解决了 GBN 协议会大量重传分组这一问题[1]。下面将详细的介绍 SR 协议的发送方和接收方的特点。

发送方和接收方维护的窗口如图 1.2 所示。SR 协议发送方维护的窗口和 GBN 协议类似，对不同事件的响应函数和 GBN 也是一致的，但是实现细节上有一定的区别，不同的是 SR 协议发送方为窗口中每一个发送但是还未确认的报文都维护了一个计时器，当该计时器超时的时候仅仅重发该计时器对应的报文，而不是像 GBN 协议一样重发所有还未收到确认的报文。

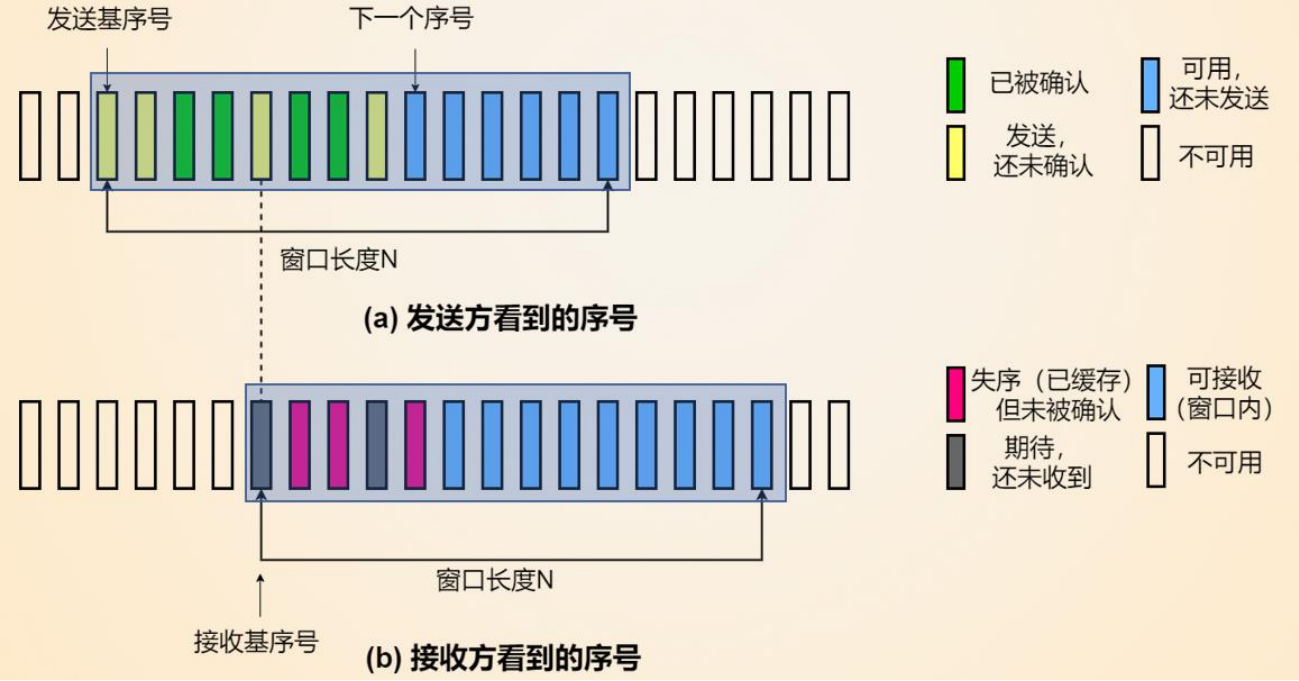

图 1.2 SR 协议窗口示意图

相比与 GBN 协议，SR 协议的接收方维护了一个窗口，用来缓存失序到达的报文段，如果收到的报文段是接收方窗口的基序号的话，就移动接收方的窗口；如果收到的报文段的序号在接收方窗口中的话，就对该报文进行确认；如果收到的报文段的序号在[base−N, base−1]的话，就重发上一个 ACK，防止发送方的窗口因为丢包无法移动。

1.3.3 TCP 协议的设计

这里我们实现的是一个比较简单的 TCP 协议，仅仅在 GBN 协议的基础上增加了拥塞避免的算法[1]。具体来说，就是在 GBN 协议的基础上增加了对上一次收到的 ACK 的序号的记录，如果重复收到三个冗余 ACK 的话，就进入拥塞避免状态，重传最早的还未收到确认的报文段

即可。

1.4 系统实现

在本次实验中，我设置的窗口大小 N 为 4，分组序号的比特数为 3，也就是分组的序号空间为[0, 7]，当然为了最终检查时看的更加清晰，我在输出分组序号的时候没有对 8 取模。由于给出的代码模板中已经实现了一个可以进行可靠数据传输的停等协议，因此这里只需要重写一个 RdtSender 类和一个 RdtReceiver 类即可。

1.4.1 GBN 协议的实现

根据前文的分析，我们可以知道 GBN 协议的发送方需要维护一个大小为 N 的窗口，整个系统中仅有一个计时器，该计时器和窗口中的第一个分组绑定，如果计时器超时的话就重发所有已发送但是未被确认的分组，如果窗口发生移动的话就重启计时器。而接收方的动作相对简单很多，仅仅可以视为维护了一个大小为 1 的窗口，如果收到了失序分组的话就将该分组丢弃即可，下面将详细介绍一下 GBN 发送方和接收方的一些动作的实现。

GBNSender::send

该函数主要用来实现发送方从上层接受数据并将其打包发送的动作，具体的逻辑如图 1.3 所示。

首先发送方将判断当前是否处于等待确认状态，其实就是判断一下 waitingState 这个量是否为 true，如果不是的话就判断当前的窗口是否还有空余位置，具体的就是用 expectSequenceNumberSend 这个量和 base+N 进行比较，如果还有空余位置的话就可以将此次从上层接收到的 msg 发送出去，需要更新一下 expectSequenceNumberSend 这个量，并且判断一下当前的窗口起点的序号对应的分组是不是已发送但是还没有收到确认的分组，如果是的话就说明需要启动计时器，此外，这里在发送后也需要判断一下窗口是否已满，进而来更新 waitingState 这个量，当然，其实 waitingState 这个量是完全不需要的，因为后续我们也会首先判断一下窗口是否已满，但是加入这个量会使我们的系统更加清晰。

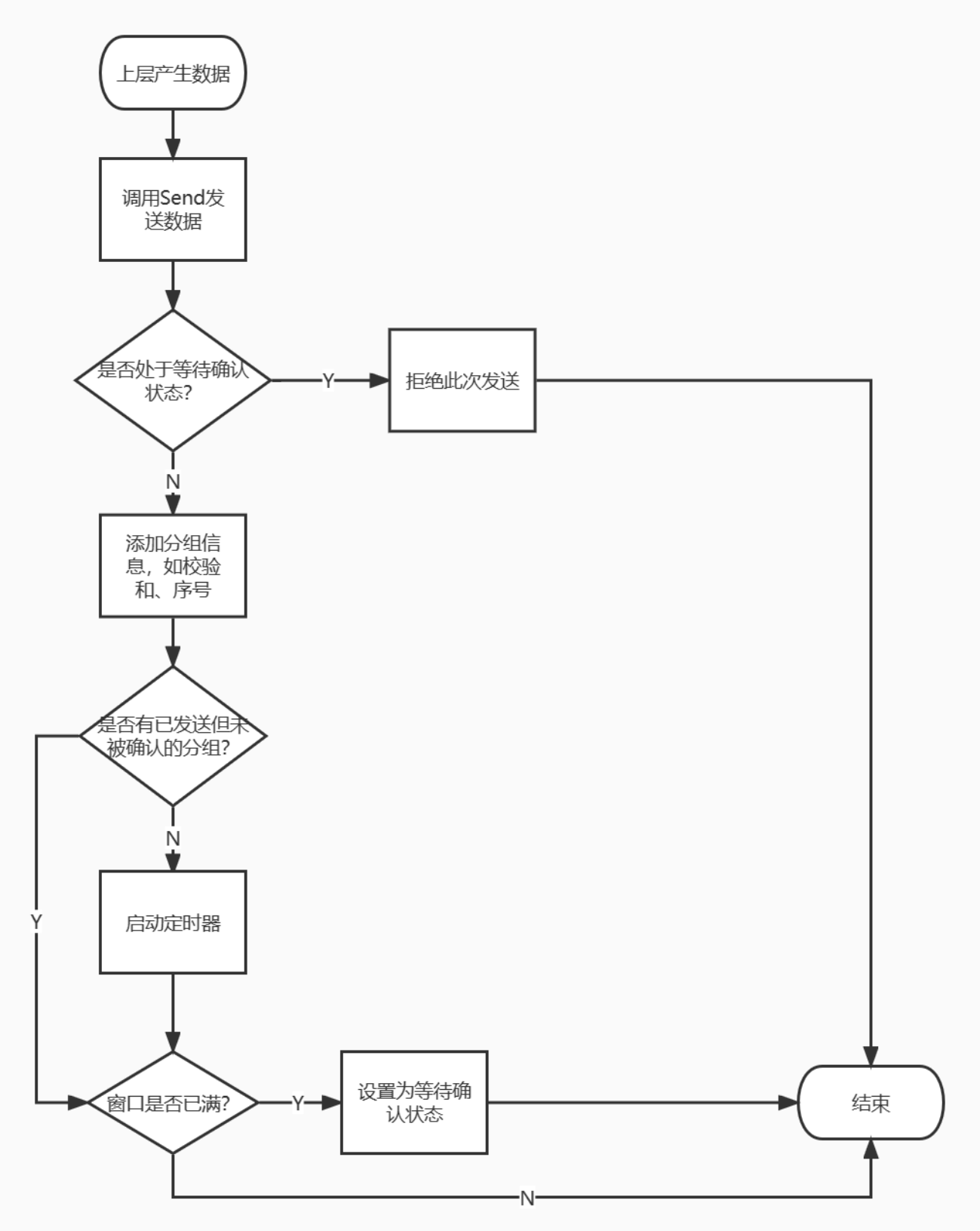

图 1.3 GBN 发送方 send 逻辑流程图

GBNSender::receive

该动作主要是 GBN 发送方对 GBN 接收方返回的 ACK 的回应，发送方通过该信息来更新

窗口的位置，具体的逻辑如图 1.4 所示。

当发送方收到接收方的 ACK 的时候，首先需要调用模拟环境给出的 calculateCheckSum 函数计算校验和，如果收到的 ACK 的校验和正确并且收到的 ACK 的序号不小于当前发送方的分组的基序号 base 的话，根据累计确认，需要移动发送方的滑动窗口，并且为了便于确认 GBN 协议运行过程的正确性，这里也会打印出当前的滑动窗口的内容，根据此时滑动窗口的基序号 base 和下一个待发送的序号 expectSequenceNumberSend 的大小关系判断此时滑动窗口中是否还有那些已发送但是还未被确认的分组，从而决定是要关闭计时器还是要重启计时器。

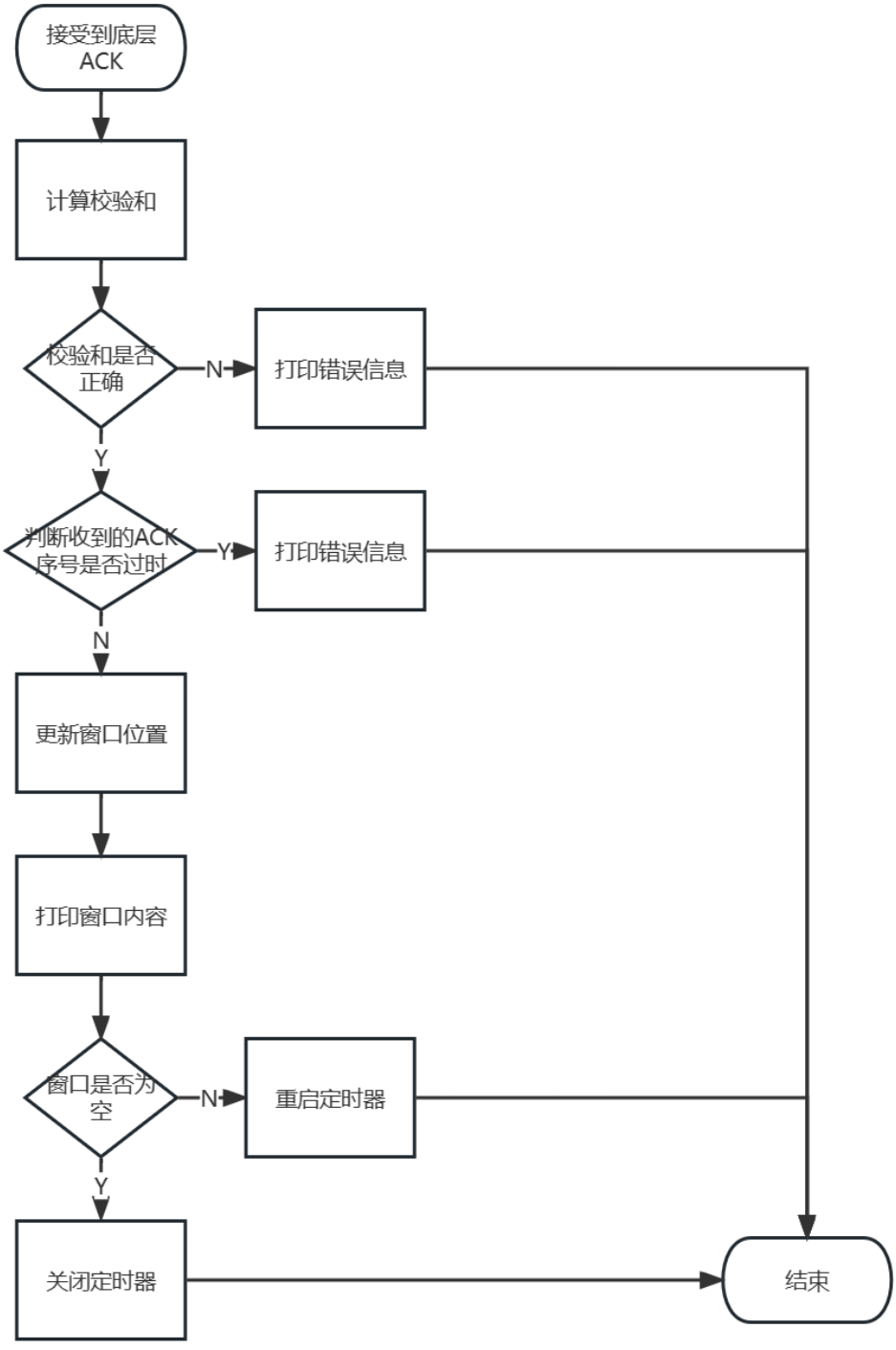

图 1.4 GBN 发送方的 receive 逻辑流程图

GBNSender::timeHandler

该函数是在发送方察觉到定时器超时的时候做出的相应反应，对于 GBN 而言，需要将已发送但是还没有收到回应的报文全部重传，由于本身该函数逻辑比较简单，所以省去了流程图。

GBNReceiver::receive

该函数是 GBN 接收方在收到发送方传递的数据时产生的动作，主要功能就是接收数据、将数据传递给上层、向发送方传递 ACK。核心逻辑如图 1.5 所示。

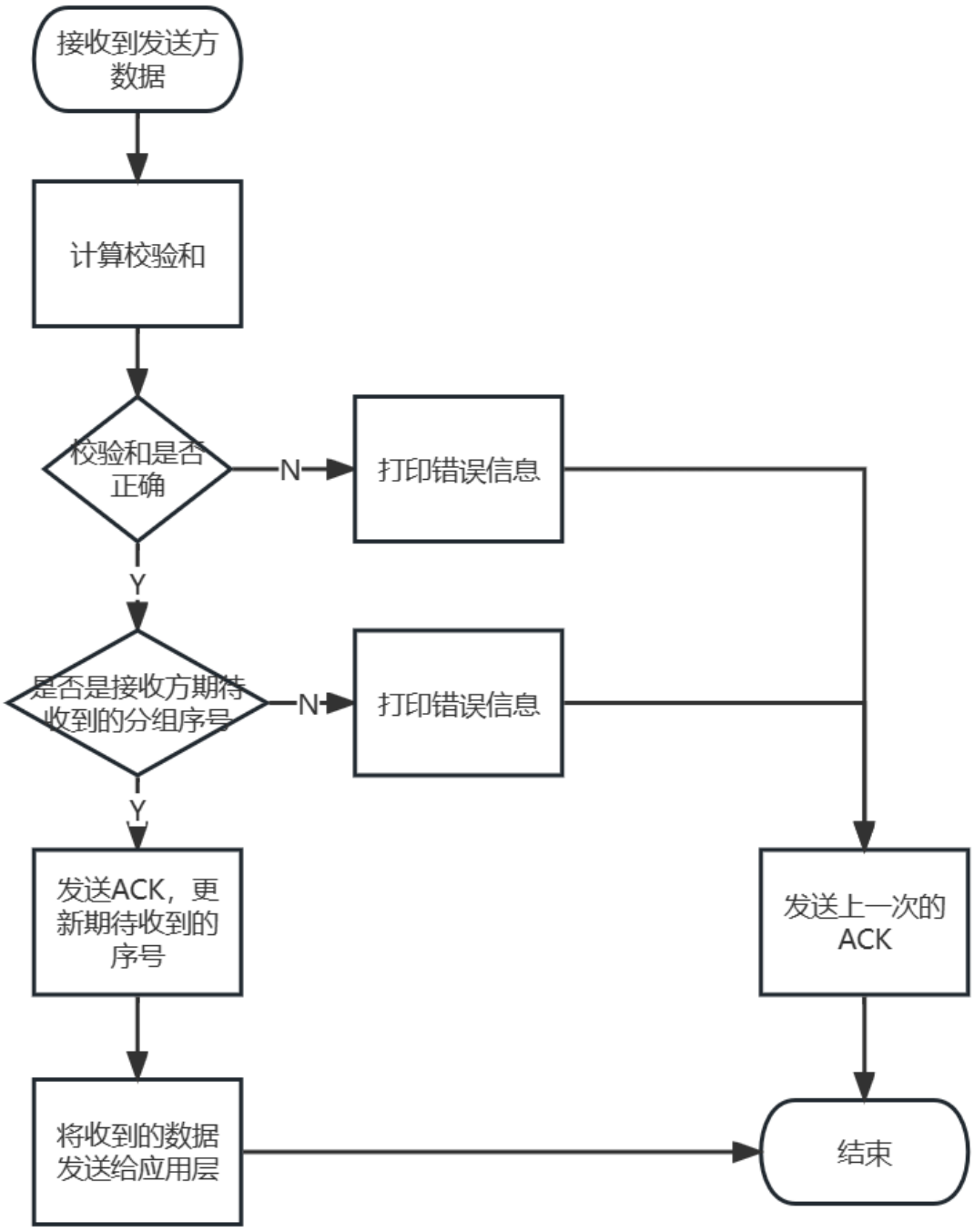

图 1.5 GBN 接收方 receive 逻辑流程图

当接收方收到发送方的分组时，首先需要计算该分组的检验和，确定该分组在传递过程中

是否受到损坏，如果没有损坏就判断一下该分组的序号和接收方当前期待接受的分组的序号是否相同，如果相同的话就可以将收到的分组传递给上层，然后向发送方发送 ACK，并且更新当前接收方期待收到的分组序号，否则打印相应的错误信息，然后发送上一次发送的 ACK 即可。

1.4.2 SR 协议的实现

前文已经提到过 SR 协议相比于 GBN 采用了选择重传的方法，从而大大减少了需要传递的报文数量，从而获取了一个比较好的效率。由于 SR 协议在接收方同样需要维护一个窗口来缓存失序到达的分组，因此这里选择直接在 SR 协议的接收方结构体中增加一个名为 CachePacket 的数组来缓存分组，此外在分组的 Packet 结构体中增加了一个布尔类型的量 ACK，来表示当前分组是否已经被确认。

由于 SR 协议和 GBN 协议都是在停止等待协议的基础上进行设计的，因此在实现上有一些类似的地方，因此对于那些相同的部分本文不再赘述。

SRSender::send

该函数是 SR 的发送方收到上层调用后发送分组的函数，在实现思路上和 GBN 发送方的 send 函数在实现上基本一致，区别是由于 SR 使用的是选择重传，所以需要为已经发送但是还没被确认的所有分组绑定一个定时器，当定时器超时的时候仅仅重发该分组即可，逻辑流程图此处便省略了。

SRSender::receive

该函数是 SR 的发送方收到底层传递过来的接收方发送的 ACK 时进行的相关的动作，其逻辑流程图如图 1.6 所示。

首先 SR 的发送方仍然需要计算收到的 ACK 的检验和，如果检验和正确的话需要根据收到的 ACK 的序号进行一系列的判定。首先如果收到的 ACK 是对发送方窗口的 base 分组进行的确认，就应该移动发送方窗口，将从 base 开始的连续的已经被确认的分组中的 ACK 字段重新改为未被确认从而被重新复用。如果收到的 ACK 是对当前窗口的 base 之后的 ACK，并且该分组还没有被确认，就关闭该分组的定时器，并且更新该分组的 ACK 字段，否则就认为收到的该 ACK 无效。值得一提的是，SR 协议和 GBN 协议均选择在发送方收到 ACK 的时候打印当前的窗口，并且 SR 协议在各种情况下均会打印相应的提示信息说明当前收到的 ACK 的状态情况，比如该 ACK 是否是无用 ACK 等等。

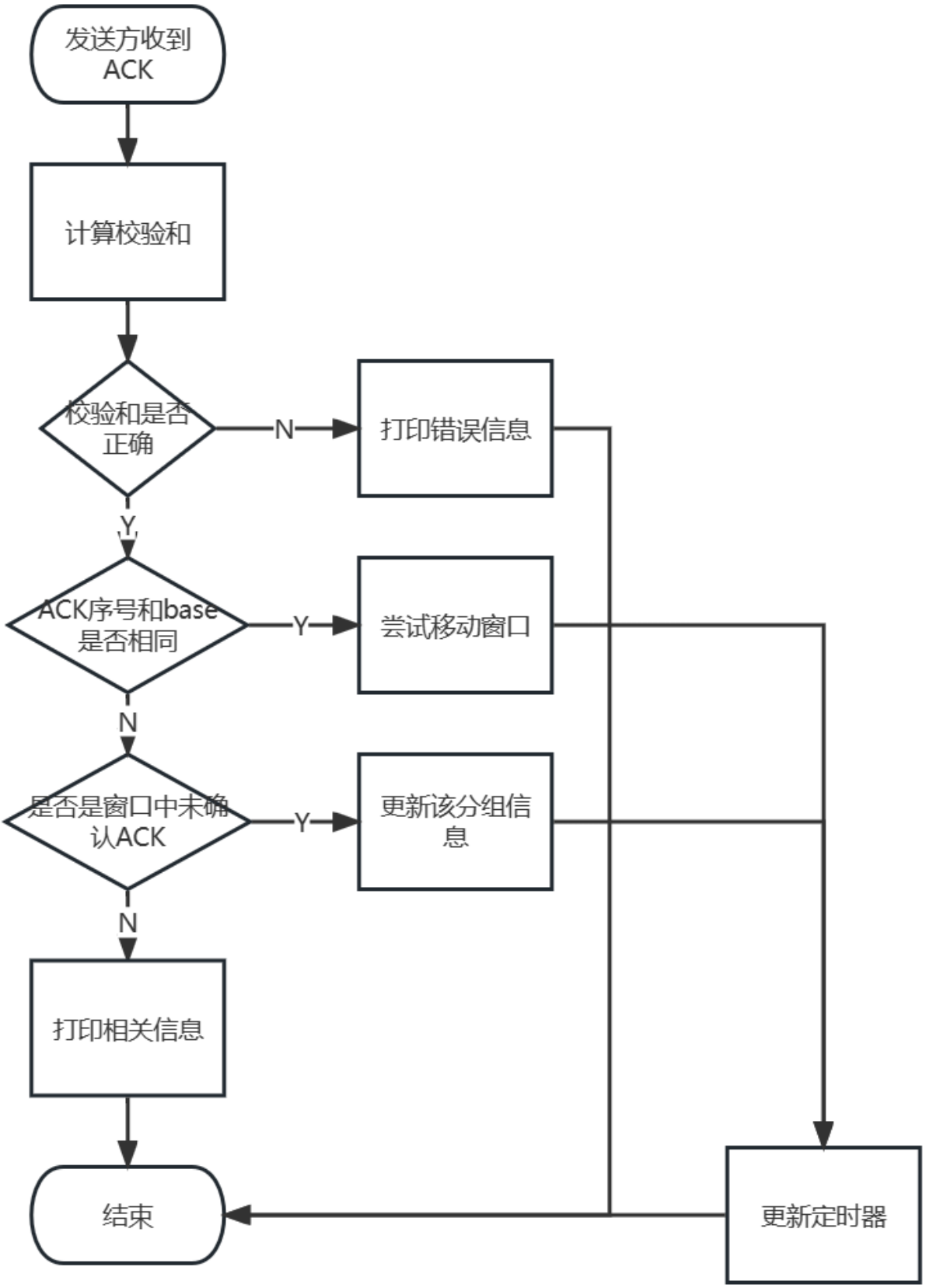

图 1.6 SR 发送方 receive 逻辑流程图

SRSender::timehandler

该函数用来在某个分组的定时器超时的时候重发该分组，所以只需要将这次超时的分组发送出去即可。

SRReceiver::receive

前文已经提到过，SR 协议接收方的动作和 GBN 协议有比较大的区别，重点在于接收方滑动窗口的移动上，逻辑流程图如图 1.7 所示。

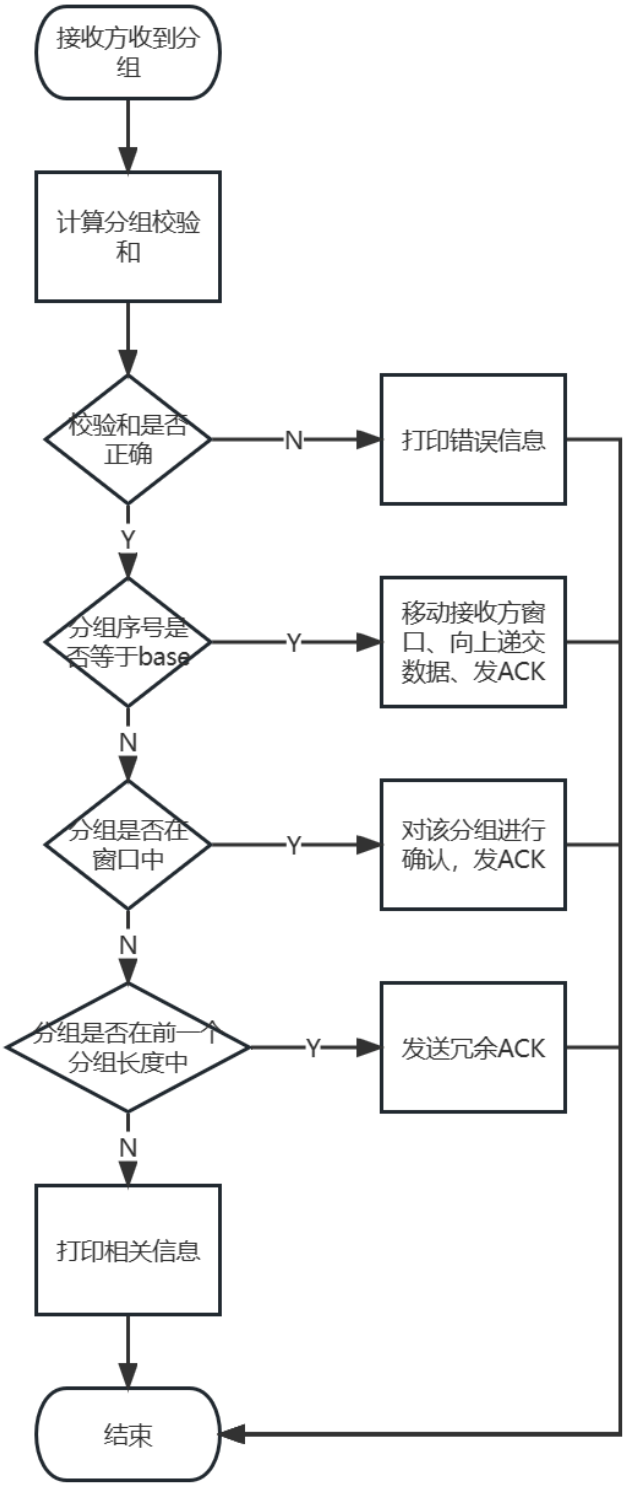

图 1.7 SR 接收方 receive 逻辑流程图

首先，接收方在 receive 的时候仍然需要先计算一下当前接收到的分组的校验和，只有在该分组的校验和正确的时候才会进行下面的动作。随后将比较一下接收到的分组的序号和接收方滑动窗口基序号，如果接收到的分组序号正好等于 base 基序号，需要对接收方的滑动窗口进行移动，需要一直移动到基序号后第一个没有被接收到的分组的位置，此外还需要对 base 序号对应的分组发送一个 ACK，同时将已经确认的连续分组递交给上层应用，这个过程的核心利用了前文提到过的在结构体中增加的 CachePacket 数组以及在分组结构中增加的 ACK 确认字段。如果接受到的分组在滑动窗口中，但是又不是 base 分组，则需要更新 CachePacket 数组中对应序号的分组的信息，同时更新 lastackPkt 这样的信息，发送对这个分组的 ACK。如果接收到的分组不再当前的滑动窗口中，说明该分组已经被确认过了，这个时候为了防止丢包造成发送方滑动窗口不能移动，接收方需要发送冗余 ACK，这也是 SR 协议的一个核心设计点。

1.4.3 TCP 协议的设计

我们这里设计的 TCP 是一个很简易的 TCP 协议，仅仅在 GBN 协议的基础上增加了拥塞避免算法，并且更改了 GBN 协议发送方在定时器超时时的行为，所以这里重点描述一下 TCP 协议和 GBN 协议不同的地方。

TCPSender::receive

相比于 GBN 发送方对 ACK 的响应，TCP 在发送方的结构体中使用了 lastack 这个信息来记录上一次收到的 ACK，如果上一个 ACK 的序号和这次收到的 ACK 序号相同的话，就认为收到了冗余 ACK，当收到三个冗余 ACK 的时候，就重启计时器，然后重发分组，其流程图如图 1.8 所示。

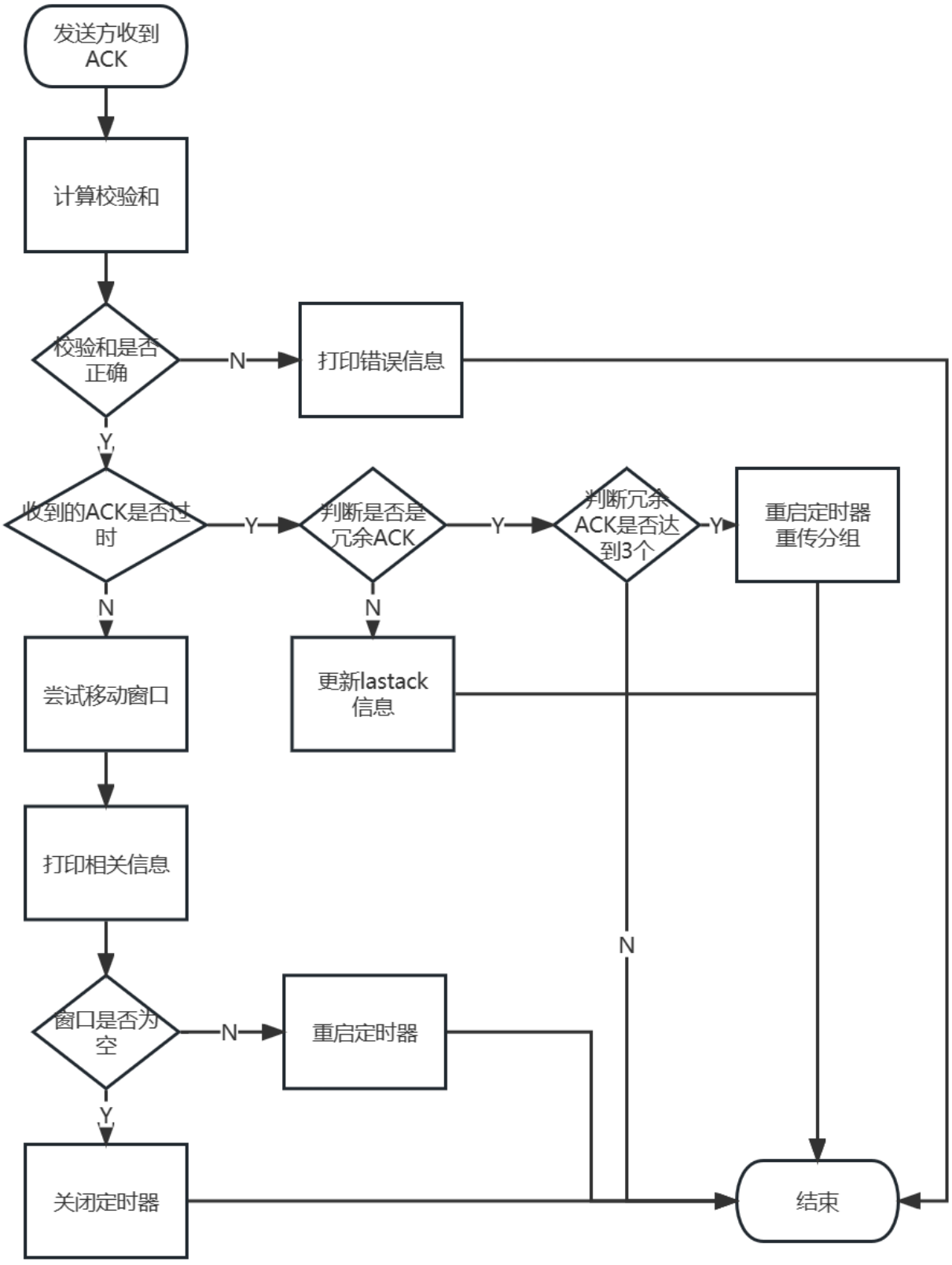

图 1.8 TCP 发送方 send 逻辑流程图

TCPSender::timeouthandler

TCP 的发送方虽然只维护了一个定时器，但是不是像 GBN 一样重传的时候将所有已发送

但是未被确认的分组全部重传，而是仅仅重传第一个已发送但是未被确认的分组。

1.5 系统测试及结果说明

硬件测试环境：

处理器：AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz

机带 RAM：16.0GB(15.4GB 可用)

测试结果与分析：

在本次实验中，需要实现 GBN、SR、TCP 三个可靠数据传输协议，因此我们测试的重点在于验证传输数据的正确性以及根据打印的日志分析程序运行过程中滑动窗口移动的正确性。这里的测试选择编写一个 Windows 下的测试脚本多次运行三个协议来验证其传输数据的正确性，并且会分析部分日志以验证滑动窗口移动的正确性。

GBN 协议的测试：

这里我们首先使用测试脚本运行 GBN 协议，运行结果如图 1.9 所示

```
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异
```

图 1.9 GBN 协议测试结果

可以说明 GBN 协议在传输可靠性上是有保证的。

接下来我们分析输出日志中的一部分，来说明 GBN 滑动窗口移动的正确性。随意截取的一段输出日志如图 1.10 所示。注意，这里由于为了便于检查，所以没有对输出的序号进行取模，所以会有像 38 这样比较大的序号。

```
发送方滑动窗口内容为 [ 38 39 40 * ]
当前分组还没接受完，继续等待
接收方正确收到发送方的报文: seqnum = 38, acknum = -1, checksum = 64213, MMMMMMMMMMMMMMMMMMMM
******模拟网络环境******：向上递交给应用层数据：MMMMMMMMMMMMMMMMMMMM
接收方发送确认报文: seqnum = -1, acknum = 38, checksum = 12813, .....
接收方没有正确收到发送方的报文,数据校验错误: seqnum = -999999, acknum = -1, checksum = 61642, NNNNNNNNNNNNNNNNNNN
接收方重新发送上次的确认报文: seqnum = -1, acknum = 38, checksum = 12813, .....
发送方收到的ACK损坏
发送方正确收到确认: seqnum = -1, acknum = 38, checksum = 12813, .....
发送方滑动窗口内容为 [ 39 40 * * ]
当前分组还没接受完，继续等待
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 40, acknum = -1, checksum = 59071, OOOOOOOOOOOOOOOOOOOO
接收方重新发送上次的确认报文: seqnum = -1, acknum = 38, checksum = 12813, .....
发送方没有收到正确的序号，继续等待
超时了，进行Back N
重发第39个报文
发送方定时器时间到，重发报文: seqnum = 39, acknum = -1, checksum = 61642, NNNNNNNNNNNNNNNNNNN
重发第40个报文
发送方定时器时间到，重发报文: seqnum = 40, acknum = -1, checksum = 59071, OOOOOOOOOOOOOOOOOOOO
接收方正确收到发送方的报文: seqnum = 39, acknum = -1, checksum = 61642, NNNNNNNNNNNNNNNNNNN
******模拟网络环境******：向上递交给应用层数据：NNNNNNNNNNNNNNNNNNN
接收方发送确认报文: seqnum = -1, acknum = 39, checksum = 12812, .....
接收方正确收到发送方的报文: seqnum = 40, acknum = -1, checksum = 59071, OOOOOOOOOOOOOOOOOOOO
******模拟网络环境******：向上递交给应用层数据：OOOOOOOOOOOOOOOOOOOO
接收方发送确认报文: seqnum = -1, acknum = 40, checksum = 12811, .....
超时了，进行Back N
重发第39个报文
发送方定时器时间到，重发报文: seqnum = 39, acknum = -1, checksum = 61642, NNNNNNNNNNNNNNNNNNN
重发第40个报文
发送方定时器时间到，重发报文: seqnum = 40, acknum = -1, checksum = 59071, OOOOOOOOOOOOOOOOOOOO
发送方正确收到确认: seqnum = -1, acknum = 39, checksum = 12812, .....
发送方滑动窗口内容为 [ 40 * * * ]
当前分组还没接受完，继续等待
发送方收到的ACK损坏
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 39, acknum = -1, checksum = 61642, NNNNNNNNNNNNNNNNNNN
接收方重新发送上次的确认报文: seqnum = -1, acknum = 40, checksum = 12811, .....
发送方正确收到确认: seqnum = -1, acknum = 40, checksum = 12811, .....
发送方滑动窗口内容为 [ * * * * ]
当前分组以全部接收，关闭计时器
```

图 1.10 GBN 协议部分日志

可以看到最开始发送方滑动窗口的内容为[38,39,40,*]，随后接收方收到了序号为 38 的报文，并且发送 ACK，并且序号为 39 的报文发生了损坏。发送方收到了对 38 的 ACK 之后滑动窗口向前移动。随后由于定时器超时，发送方将当前已经发送但是没有收到 ACK 的分组全部重发，这里就是第 39 和第 40 号分组，随后应该是由于接收方发送的 ACK 丢了，所以再次超时了，需要进行重传。

SR 协议测试与分析：

这里我们首先使用测试脚本运行 SR 协议，运行结果如图 1.11 所示。

```
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异
```

图 1.11 SR 协议的测试

接下来我们简单分析一下 SR 协议打印的一部分日志，如图 1.12 所示。

```
发送方滑动窗口内容为 [ 1 2 3 4 ]
收到的该报文的ack序号为1
接收方没有正确收到发送方的报文,数据校验错误: seqnum = -999999, acknum = -1, checksum = 21843, DDDDDDDDDDDDDDDDDDDD
接收方滑动窗口内容为：[ 3 4 5 6 ]
接收方收到的报文序号为4
接收方已缓存发送方的报文: seqnum = 4, acknum = -1, checksum = 19272, EEEEEEEEEEEEEEEEEEE
接收方发送确认报文: seqnum = -1, acknum = 4, checksum = 12847, .....
超时了
重发第2个报文
发送方定时器时间到，重发报文: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
超时了
重发第3个报文
发送方定时器时间到，重发报文: seqnum = 3, acknum = -1, checksum = 21843, DDDDDDDDDDDDDDDDDDDD
发送方正确收到确认: seqnum = -1, acknum = 2, checksum = 12849, .....
发送方滑动窗口内容为 [ 2 3 4 * ]
收到的该报文的ack序号为2
超时了
重发第4个报文
发送方定时器时间到，重发报文: seqnum = 4, acknum = -1, checksum = 19272, EEEEEEEEEEEEEEEEEEE
接收方滑动窗口内容为：[ 3 4 5 6 ]
接收方正确收到已确认的过时报文: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
接收方发送确认报文: seqnum = -1, acknum = 2, checksum = 12849, .....
发送方正确收到确认: seqnum = -1, acknum = 4, checksum = 12847, .....
发送方滑动窗口内容为 [ 3 4 * * ]
收到了窗口中的未确认ack4
接收方滑动窗口内容为：[ 3 4 5 6 ]
接收方收到的报文序号为基序号3
接收方发送确认报文: seqnum = -1, acknum = 3, checksum = 12848, .....
******模拟网络环境******：向上递交给应用层数据：DDDDDDDDDDDDDDDDDDDD
******模拟网络环境******：向上递交给应用层数据：EEEEEEEEEEEEEEEEEEE
发送方正确收到确认: seqnum = -1, acknum = 2, checksum = 12849, .....
发送方滑动窗口内容为 [ 3 4 * * ]
```

图 1.12 SR 协议部分日志

从这部分日志中可以看到 SR 协议和 GBN 协议的明显区别，也就是对于每一个报文都维护了一个定时器，当定时器超时的时候仅仅重发该超时事件对应的报文。例如在滑动窗口内容为[2,3,4,*]的时候，4 号分组发生了超时事件，因此仅仅重传 4 号分组。

TCP 协议测试与分析：

这里我们首先使用测试脚本运行 TCP 协议，运行结果如图 1.13 所示。

```
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异
```

图 1.13 TCP 协议测试结果

然后我们简单分析一下 TCP 协议的日志，这里重点关注 TCP 中拥塞控制部分的日志，如图 1.14 所示。

```
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 86, acknum = -1, checksum = 8910, IIIIIIIIIIIIIIIIIII
接收方重新发送上次的确认报文: seqnum = -1, acknum = 84, checksum = 12767, .....
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 87, acknum = -1, checksum = 6339, JJJJJJJJJJJJJJJJJJ
接收方重新发送上次的确认报文: seqnum = -1, acknum = 84, checksum = 12767, .....
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 88, acknum = -1, checksum = 3768, KKKKKKKKKKKKKKKKKKKK
接收方重新发送上次的确认报文: seqnum = -1, acknum = 84, checksum = 12767, .....
超时了，进行Back N
重发第85个报文
发送方定时器时间到，重发报文: seqnum = 85, acknum = -1, checksum = 11481, HHHHHHHHHHHHHHHHHHHH
接收方没有正确收到发送方的报文,数据校验错误: seqnum = 85, acknum = -1, checksum = 11481, IHHHHHHHHHHHHHHHHHHH
接收方重新发送上次的确认报文: seqnum = -1, acknum = 84, checksum = 12767, .....
收到了3个冗余ack，开始快速重传，冗余ack序号为84
```

图 1.14 TCP 协议部分日志

可以看到这里发送方共收到了 4 个序号为 84 的 ACK，然后便认为网络中可能发生拥塞，进行快速重传。

1.6 其它需要说明的问题

无

1.7 参考文献

[1] James F·Kurose / Keith W. Ross（美）. 计算机网络：自顶向下方法（原书第 8 版）. 北京：机械工业出版社.

心得体会与建议

2.1 心得体会

在这次计算机网络的实验课中，我通过不断地学习对理论课上学到的应用层、运输层、网络层、链路层的知识有了更加深刻的理解，在实践中对计算机网络的体系结构有了更深刻的理解。

在实验一 socket 编程中，我通过 socket 相关的函数实现了一个简易的 webserver，由于之前基本没有接触到过用 C++进行网络编程相关的知识，所以这个实验在最开始上手的时候还是有一定难度的，但是实际上由于老师给的实验指导书非常详细，并且最开始给出的 demo 部分也比较完备，所以最后也没有出现很大的问题。通过这个实验，我对计算机网络应用层中 http 协议相关的知识有了更深的理解，对 web 界面的请求过程有了更深层次的理解。

在实验二可靠数据传输中，我在最基本的停等协议的基础上实现了 GBN、SR、TCP 三个流水线化的可靠数据传输协议，由于这部分是理论课上学习的重点，并且实验指导书相关的部分也非常的详细，所以这部分其实没有遇到很大的问题，只要理解了待实现的这三个协议的区别和联系，就可以比较快的完成这个实验。

在实验三 CPT 组网实验中，本身涉及到的知识相对来说是比较简单的，只需要根据实验文档学习一下软件的用法即可，同时重点其实是跟着任务书中给出的例子学习一下路由器的几个配置。

总的来说，通过这三次实验，我对计算机网络的相关知识有了更深层次的理解。

2.2 建议

个人建议实验课可以开的早一点，感觉和理论课的时间差的太远了。